代数的仕様記述における詳細化 : 特に抽象的順序機械の場合 (数理情報科学の基礎理論と応用)

AUTHOR(S):

鈴木, 一郎; 杉山, 裕二; 谷口, 健一; 嵩, 忠雄

CITATION:

鈴木, 一郎 ...[et al]. 代数的仕様記述における詳細化 : 特に抽象的順序機械の場合 (数理情報科学の基礎理論と応用). 数理解析研究所講究録 1981, 421: 92-105

ISSUE DATE:

1981-03

URL:

http://hdl.handle.net/2433/102548

数理解析研究所講究録
421巻 1981年 92-105

92

代数的仕様記述における詳細化
—特に抽象的順序機械の場合—

大阪大学基礎工学部　鈴木　一郎　杉山　裕二
谷口　健一　嵩　忠雄

基底代数を前提とする代数的仕様記述，特に抽象的順序機械による関数の実現について，一般的な理論的枠組みを与え，レコードの書込み，読出しに時間のかかるディスクに関連した具体例を示す．

1. 代数的仕様

以下，基本的な定義は(1)，(2)等に従う．仕様$\mathcal{S}$は(S,Σ,E)で与えられる．S，Σ，Eはそれぞれソート，関数記号，公理の集合である．以下では，各公理$\in E$の左辺には同じ変数は複数回現れず，右辺に現れる変数は必ず左辺に現れているものとする．

基底代数Bは仕様$spec(B)=(S_B,\Sigma_B,E_B)$で表されるとする．[2] $spec(B)$には代数Bの集合の元に一対一に対応する定数関数（B-定数と呼ぶ）があり，公理$\in E_B$は（一般には無限の）

関数の定義表として与えられる．また，二つのB-定数が等しいかどうかを判定するための関数"="が各ソートにある．仕様$\mathfrak{D}=(S, \Sigma, E)$がspec(B)を部分仕様として含むとき，B-仕様であるという．どの二つの異なるB-定数b_1, b_2に対しても$b_1 \equiv_{\mathfrak{D}} b_2$とならないとき，$\mathfrak{D}$は無矛盾であるという（$\equiv_{\mathfrak{D}}$は$\mathfrak{D}$の公理系によって与えられる（$\Sigma$上の項の集合$T_\Sigma$の上の）合同関係を表す）．

B-仕様 $\mathfrak{D}=(S_B + S,\ \Sigma_B + \Sigma_0 + \Sigma_h,\ E_B + E)$，

B'-仕様 $\mathfrak{D}'=(S_{B'} + S',\ \Sigma_{B'} + \Sigma_0' + \Sigma_h',\ E_{B'} + E')$

がそれぞれ無矛盾であり，$B \subseteq B'$（BはB'の部分代数，同型は同一視），$\Sigma_0 \subseteq \Sigma_0' \subseteq \Sigma_0 + \Sigma_h$とする（+は共通部分のない集合の和を表す）．値域がS_Bのソートである$\Sigma_B + \Sigma_0$上の任意の項t_1，t_2に対し，条件

$$「\ t_1 \equiv_{\mathfrak{D}} t_2 \ \Rightarrow\ t_1 \equiv_{\mathfrak{D}'} t_2\ 」$$

が満たされるとき，$\mathfrak{D}'$は$\mathfrak{D}$の（Σ_0に着目した）詳細化であるという．

この考えは，仕様が表すものをその始代数に限定せず，また，拡張（enrichment）におけるように，すべての項の値がB-定数として定まることを要求しない．さらに，いわゆる符号化，複号化による同型を用いた考え[3]とも異なる．ここでは，対応自身も仕様$\mathfrak{D}'$に含めて考えている．この詳細化は推移律

が成立つ．同様の枠組みで基底代数をパラメタ化できる．

2．抽象的順序機械

無矛盾なB-仕様$\mathcal{B}=(S_B+S, \Sigma_B+\Sigma, \mathcal{E}_B+\mathcal{E})$が次の条件(i), (ii)を満たすとき，特に$\mathcal{B}$を抽象的順序機械と呼ぶ．(2)

(i) Sはただ一つのソートからなる．そのソートをstateとする．

(ii) どの関数についても，引数ソートの系列中にソートstateは高々一回しか現れない．

さらに$\mathcal{B}$が条件

(iii) その公理の左辺は$f(x_1, \cdots)$又は$f'(g(x_1, \cdots), y_1, \cdots)$の形をしている．ここで，$x_1, \cdots, y_1, \cdots$は変数又は定数記号，$g$はソートがstateの関数記号．

(iv) $\mathcal{E}_B+\mathcal{E}$の各公理の左辺が，どの公理の左辺とも，又，自分自身のどの真部分項とも重なり得ない（いわゆるChurch-Rosserの性質の十分条件）．

を満たす場合は，

① 公理の形が状態遷移に直接対応しており，直感的にわかりやすい．

② 構造的帰納法を用いることが容易である．

③ $\mathcal{B}$が無矛盾であることを容易に示すことができる．

の性質がある．

表，プッシュダウンストア，"CPU"，HDLCのプロトコル，ファイル管理システム等は抽象的順序機械として記述できる．

3. 抽象的順序機械による関数の実現

$M = (S_B + \{state\},\ \Sigma_B + \Sigma_M,\ \varepsilon_B + \varepsilon_M)$ を抽象的順序機械とする．以下簡単のため，値域がソート state である各（状態遷移）関数記号

$$g_j : state, s_{j_1}, s_{j_2}, \cdots \rightarrow state \quad \in \Sigma_M$$

に対し，"$g_j(\cdots) == \cdots$" なる形の公理は ε_M に存在しないようなMのみを考える．上の各 g_j に対し，記号の集合

$$G_j = \{ g_j(b_{j_1}, b_{j_2}, \cdots) \mid b_{j_1}, b_{j_2}, \cdots \text{はそれぞれソート } s_{j_1}, s_{j_2}, \cdots \text{の B-定数} \}$$

を考え，さらにすべての g_j に対するそれらの集合の和

$$\bigcup_{g_j \in \Sigma_M \text{の状態遷移関数記号}} G_j$$

を考え，そのソートを P と書く．P の元の系列全体の集合を考え，そのソートを P^* と書く．連接 "・" はすべての系列について共通であるとする．Mの各非遷移関数記号

$$f_i : state, s_{i_1}, s_{i_2}, \cdots \rightarrow s_i \quad \in \Sigma_M$$

$(s_{i_1}, s_{i_2}, \cdots, s_i \in S_B)$ に対し，P^* 上の関数記号

$$\tilde{f}_i : P^*, s_{i_1}, s_{i_2}, \cdots \rightarrow s_i$$

を考え，Σ_Mの各公理に現れるすべての

$f_i(s, \cdots)$ を $\tilde{f}_i(\alpha, \cdots)$ に，

$g_j(s, \cdots)$ を $g_j(\cdots)\cdot\alpha$ ・・・(*) に

それぞれ対応させておきかえることにより得られるB'-仕様を$\tilde{M}$と書く（基底代数B'は，BにP, P', "・", head, tailを追加したものである）．

このとき，任意の

f_i : state, s_{i_1}, s_{i_2}, ・・・ $\rightarrow s_i$

と，対応する

$\tilde{f}_i$: P^*, s_{i_1}, s_{i_2}, ・・・ $\rightarrow s_i$

について，(*)の関係で対応するソート stateの任意の項 s と P^*の系列 α，及び任意のB-定数 $b_{i_1}, b_{i_2}, \cdots, b_i$（それぞれソート $s_{i_1}, s_{i_2}, \cdots, s_i$）に対し，

$f_i(s, b_{i_1}, b_{i_2}, \cdots) \underset{M}{\equiv} b_i \Leftrightarrow \tilde{f}_i(\alpha, b_{i_1}, b_{i_2}, \cdots) \underset{\tilde{M}}{\equiv} b_i$

が成立する．以下では記法上の簡単のため，Mのかわりに$\tilde{M}$を用いて議論する．

Pを $P_I + P_E$ のように分ける．P_Iはユーザが直接コントロールできる状態遷移（例えば機械の一命令の実行，要求の受付け）を表し，P_Eは受付けた要求に対する完了信号や割込み等の，システム内での "ε-move" を表す．

$\tilde{M}$に非遷移関数

M-control : $P^* \rightarrow$ bool

がある．M-controlは次の［条件 false. preserve］を満たさねばならない．

［条件 false. preserve］:

(i) M-control(null) $\underset{\tilde{M}}{\equiv}$ true （nullは空系列），かつ

(ii) M-control($p \cdot \alpha$) $\underset{\tilde{M}}{\equiv}$ true $\Rightarrow$ M-control(α) $\underset{\tilde{M}}{\equiv}$ true

（$p \in P$, $\alpha \in P^*$）．

M-control(α) は，初期状態から一連の状態遷移 α が機械として起りうるとき，かつそのときのみ真となるような述語である．具体的には，条件付ジャンプ命令の実行のあとでは，条件に従って決ったもののしか実行されないとか，依頼していないリクエストに対する完了信号が到来しないとかいったことを記述する．

ある B_0-仕様 $\tilde{M}_0$ で，集合（ソート）P_0, P_0^* と二つの関数

$f_0 : P_0^*, S_{01}, S_{02}, \cdots \rightarrow S_0$

$wd_0 : P_0^*, S_{01}, S_{02}, \cdots \rightarrow$ bool

が与えられたとする（P_0, P_0^*, S_{01}, S_{02}, …, S_0 は B_0-ソート）．wd_0 は次の性質

$wd_0(p \cdot x) \underset{\tilde{M}_0}{\equiv}$ true $\Rightarrow$ $wd_0(x) \underset{\tilde{M}_0}{\equiv}$ true （$p \in P_0$, $x \in P_0^*$）

をもつ．

ここで，次のような仕様 m を考える（図1参照）．

(1) $\mathcal{M}$ は無矛盾.

(2) $\mathcal{M}$ にソート P_0, P_0^*, 関数記号 f_0 がある.

(3) $\mathcal{M}$ は $\tilde{M}$ を部分仕様としてもつ.

(4) $\mathcal{M}$ に非遷移関数

$$\text{Interpret} : P_0^*, P^* \rightarrow \text{bool}$$

がある. Interpret は次の条件 (1) ~ (6) を満たす ($p \in P_0$, $x \in P_0^*$, $y, y_1 \in P^*$).

(1) $\text{Interpret}(\text{null}, \text{null}) \underset{\mathcal{M}}{\equiv} \text{true}$

(2) $\text{Interpret}(x, y) \underset{\mathcal{M}}{\equiv} \text{true}$ かつ $wd_0(p \cdot x) \underset{\tilde{M}_0}{\equiv} \text{true}$

$\Rightarrow (\exists y' \in P^*)(\text{Interpret}(p \cdot x, y'y) \underset{\mathcal{M}}{\equiv} \text{true})$

(3) $\text{Interpret}(x, y) \underset{\mathcal{M}}{\equiv} \text{true}$

$\Rightarrow wd_0(x) \underset{\tilde{M}_0}{\equiv} \text{true}$ かつ $\text{M-control}(y) \underset{\tilde{M}}{\equiv} \text{true}$

(4) $\text{Interpret}(x, y) \underset{\mathcal{M}}{\equiv} \text{true}$ かつ $\text{Interpret}(x, y'y) \underset{\mathcal{M}}{\equiv} \text{true}$

$\Rightarrow y' \in P_E^*$

(5) $\text{Interpret}(x, y_1 y) \underset{\mathcal{M}}{\equiv} \text{true}$ かつ $y_1 \in P_E^*$

$\Rightarrow \text{Interpret}(x, y) \underset{\mathcal{M}}{\equiv} \text{true}$

(6) $\text{Interpret}(x, y_1 y) \underset{\mathcal{M}}{\equiv} \text{true}$

かつ $g \in P_E$ かつ $\text{M-control}(g \cdot y) \underset{\tilde{M}}{\equiv} \text{true}$

$\Rightarrow (\exists y'' \in P^*)(\text{Interpret}(x, y''g y) \underset{\mathcal{M}}{\equiv} \text{true})$.

Interpret (α, β) は, 上のレベルの系列 $\alpha \in P_0^*$ に対し, 順序機械 $\tilde{M}$ で状態遷移 $\beta \in P^*$ が可能であり, しかも α に対して今

後起りうる $\tilde{M}$ の遷移は P_E で表される "ε-move" のみである（条件(4)）か否かを示す述語であり，機械 $\tilde{M}$ の可能な状態遷移（M-control）をさらに制限する（条件(3)）"プログラム" を表すと考えられる．条件(1)，(2)は上のレベルで意味のある（wd_0 が真）系列に対して $\tilde{M}$ で必ず "実行" できることを示す．条件(5)，(6)は P_E の "ε-move" をプログラムでコントロールできないことを表す．

f_0 に対し，$\tilde{M}$ に関数

$$f : P^*, s_{01}, s_{02}, \cdots \rightarrow s_0$$

があり，任意の B-定数 $b_{01}, b_{02}, \cdots$（それぞれソート $s_{01}, s_{02}, \cdots$）と B-定数 $\alpha \in P_0^*$，$\beta \in P^*$ について，

$$\text{Interpret}(\alpha, \beta) \underset{\mathfrak{M}}{\equiv} \text{true}$$

$$\Rightarrow f_0(\alpha, b_{01}, b_{02}, \cdots) \underset{\mathfrak{M}}{\equiv} f(\beta, b_{01}, b_{02}, \cdots)$$

が成立するとき，「f_0 は wd_0 のもとで，$\mathfrak{M}$ において $\tilde{M}$ により定義される」という．さらに $\mathfrak{M}$ が（f_0 に着目した）$\tilde{M}_0$ の詳細化になっているとき，「$\tilde{M}_0$ の f_0 は wd_0 のもとで，$\mathfrak{M}$ において $\tilde{M}$ により実現される」という．

Interpret による対応の概念は，"derivor" による詳細化や同型写像を用いる方法よりも一般的で，"ε-move" を含んだ場合のように，これらの枠組みが必ずしも適当とは思われない例についても，自然な記述が行なえる．

4. 例

各基底代数がもつソート（集合）を図2に示す．Spec Table（図3）は抽象的な"表"を表す．表の検索項目となる値の集合がaddress，表に書かれる値の集合がrecordである．関数accessは，表に書かれたrecordの値を出力するが，指定されたaddressの値で以前にrecordが書かれていなければ，特別なrecordの元，nullrecordをその値とする．Spec AbstractMemory（図4）は3.の$\tilde{M}_0$に対応し，P1がP_0にあたる．関数contentはf_0に対応し，最後のGET(d)で指定されたaddress dで表を検索した結果のrecordを値とする．ここではcontentは全域関数であるので，wd_0は省略されている．Spec Disksystem（図5）は3.の$\tilde{M}$にあたり，Tableとしてのディスクと，書込み用バッファ（WriteBuffer），読み出し用バッファ（ReadBuffer）を用いて，レコードの読み出し，書込みに時間がかかる場合を記述している．P2が3.のPにあたり，P2のうちP_E = {Complete}，他はP_Iである．書込み要求ReqW(d, r)によりレコードrがWriteBufferに記憶され，完了のε-move Completeが起こるとディスクのd番地の値として定義される．読み出しも同様であり，ReqR(d)に対しCompleteが起こると，ディスクのd番地のレコードの値が，ReadBufferの値として定義される．TJは要求に対する仕事が完了したかどうかを判定するための命令，

ℓ : Busy フラグ (Busy) をテストせよ．もし Busy = true
なら goto ℓ，そうでなければ次の命令へ．

が 1 回実行されることを示し，Complete による状態遷移 (Busy が false になる) が起こるまでくり返される．

Spec Definition By Disksystem (図 6) は 3. の $\mathfrak{M}$ にあたり，

① AbstractMemory, Definition By Disksystem 共に無矛盾．

② Interpret は 3. の条件 (1) ～ (6) を満たす．

③ 任意の $\alpha \in P1^*$, $\beta \in P2^*$ に対し，

$$\text{Interpret}(\alpha, \beta) \underset{\text{Definition By Disksystem}}{\equiv} \text{true}$$

$\Rightarrow$ ($\exists$ B-定数 $r \in$ record)

$$\left(\text{content}(\alpha) \underset{\text{Abstract Memory}}{\equiv} r \quad \text{かつ} \right.$$

$$\left. \text{content}(\beta) \underset{\text{Definition By Disksystem}}{\equiv} r \right).$$

が成立する (証明は略す). 従って，3. の定義に従って，

「Abstract Memory の content は，Definition By Disksystem において，Disksystem により実現される」

となる．

文献

(1) 杉山，谷口，嵩："代数的記述言語とその部分言語としての関数的プログラミング言語"，信学技報，AL 79 - 99 (1980-01).

(2) ——："基底代数を前提とする代数的仕様記述"，信学論 (採

録決定).

(3) Ehrich and Lipeck:"Proving Implementations Correct - Two Alternate Approaches",Information Processing 80.

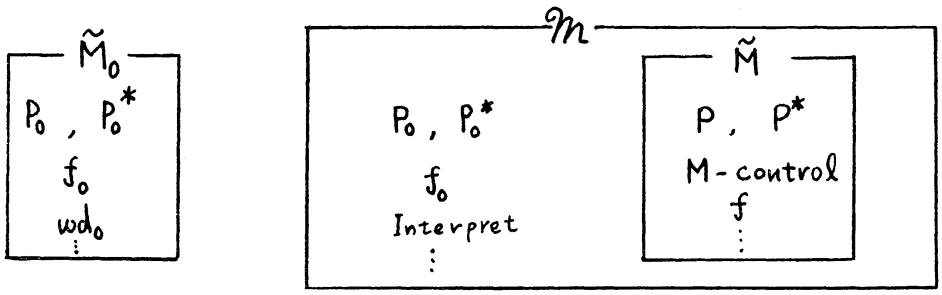

図 1. $\tilde{M}_0$ と $\mathcal{M}$ の構造

```
Algebra0: address, record, bool,
          P0 = {PUT(d,r) | d is a B-constant of sort address,
                           r is a B-constant of sort record},
          P0*
Algebra1: all sorts in Algebra0,
          P1 = {PUT(d,r) | d is a B-constant of sort address,
                           r is a B-constant of sort record}
             + {GET(d) | d is a B-constant of sort address},
          P1*
Algebra2: all sorts in Algebra0,
          type = {"WRITE", "READ"},
          P2 = {ReqW(d,r) | d is a B-constant of sort address,
                            r is a B-constant of sort record}
             + {ReqR(d) | d is a B-constant of sort address}
             + {Complete,TJ},
          P2*
Algebra3: all sorts in Algebra1,Algebra2
```

図 2. LIST OF SORTS IN BASE ALGEBRAS

```
SPEC Table;

 BASE Algebra0;
 OP
   access : P0*, address  ->  record;
 VAR
   x     SORT P0*;
   d,d´  SORT address;
   r     SORT record;
 AX
   A1:   access(null,d) == nullrecord;
   A2:   access(PUT(d,r).x, d´)
         == if d=d´ then r else access(x,d´);

END;
```

図 3. Spec Table

```
SPEC AbstractMemory;

 BASE Algebra1;
 INCLUDE Table;
 OP
   eraseGET : P1*   ->  P0*;
   content  : P1*   ->  record;
 VAR
   x  SORT P1*;
   d  SORT address;
   r  SORT record;
 AX
   EG1:  eraseGET(null) == null;
   EG2:  eraseGET(PUT(d,r).x) == PUT(d,r).eraseGET(x);
   EG3:  eraseGET(GET(d).x) == eraseGET(x);
   C1:   content(null) == nullrecord;
   C2:   content(PUT(d,r).x) == content(x);
   C3:   content(GET(d).x) == access(eraseGET(x),d);

END;
```

図4. Spec AbstractMemory

```
SPEC Disksystem;
 BASE Algebra2;
 INCLUDE Table;
 OP
   ReadBuffer      :  P2*  ->  record;
   WriteBuffer     :  P2*  ->  record;
   Disk            :  P2*  ->  P0*;
   RequestType     :  P2*  ->  type;
   AddressRegister :  P2*  ->  address;
   Busy            :  P2*  ->  bool;

   M-control       :  P2*  ->  bool;
   Testing         :  P2*  ->  bool;

 VAR
   x  SORT P2*;
   d  SORT address;
   r  SORT record;

 AX
   MC1:  M-control(null) == true;
   MC2:  FOR s = ReqW(d,r).x AND ReqR(d).x
         M-control(s) ==
         == if M-control(x) then not Testing(x);
            else false;
   MC3:  M-control(Complete.x)
         == if M-control(x) then Busy(x)
            else false;
   MC4:  M-control(TJ.x) == M-control(x);

   I1:   ReadBuffer(null) == nullrecord;
   I2:   Disk(null) == null;
   I3:   Busy(null) == false;
   I4:   Testing(null) == false;
```

図5. Spec Disksystem

```
RW:   LET s = ReqW(d,r).x IN
      ReadBuffer(s) == if M-control(s) then ReadBuffer(x),
      WriteBuffer(s) == if M-control(s) then
                          if not Busy(x) then r,
      Disk(s) == if M-control(s) then Disk(x),
      RequestType(s) == if M-control(s) then
                          if not Busy(x) then "WRITE",
      AddressRegister(s) == if M-control(s) then
                              if not Busy(x) then d,
      Busy(s) == if M-control(s) then true,
      Testing(s) == if M-control(s) then Testing(x);

RR:   LET s = ReqR(d).x IN
      FOR G IN {ReadBuffer, WriteBuffer, Disk}
      G(s) == if M-control(s) then G(x),
      RequestType(s) == if M-control(s) then
                          if not Busy(x) then "READ",
      AddressRegister(s) == if M-control(s) then
                              if not Busy(x) then d,
      Busy(s) == if M-control(s) then true,
      Testing(s) == if M-control(s) then Testing(x);

CPLT: LET s = Complete.x IN
      ReadBuffer(s)
      == if M-control(s) then
           if Busy(x) then
             if RequestType(x) = "READ" then
               access(Disk(x),AddressRegister(x))
             else ReadBuffer(x),
      WriteBuffer(s) == if M-control(s) then WriteBuffer(x),
      Disk(s)
      == if M-control(s) then
           if Busy(x) then
             if RequestType(x) = "WRITE" then
               PUT(AddressRegister(x),WriteBuffer(x)).Disk(x)
             else Disk(x),
      Busy(s) == if M-control(s) then false,
      Testing(s) == if M-control(s) then Testing(x);

TJ:   LET s = TJ.x IN
      FOR G IN {ReadBuffer, WriteBuffer, Disk,
                RequestType, AddressRegister}
      G(s) == if M-control(s) then G(x),
      FOR G IN {Busy, Testing}
      G(s) == if M-control(s) then Busy(x);

END;
```

図 5. (続き)

```
SPEC Definition By Disksystem;

 BASE Algebra3;
 INCLUDE Disksystem;
 OP
   content     :  P1*        ->  record;
   typical     :  P1*        ->  P2*;

   P-control   :  P2*        ->  bool;
   Requesting  :  P2*        ->  bool;
   Interpret   :  P1*, P2*   ->  bool;
   Extract     :  P2*        ->  P1*;
 VAR
   x  SORT  P1*;
   y  SORT  P2*;
   d  SORT  address;
   r  SORT  record;
 AX
   CON1: content(x) == ReadBuffer(typical(x));

   TYP1: typical(null) == null;
   TYP2: typical(PUT(d,r).x) == Complete.ReqW(d,r).typical(x);
   TYP3: typical(GET(d).x) == Complete.ReqW(d).typical(x);

   PC1:  P-control(null) == true;
   PC2:  FOR s = ReqW(d,r).y AND ReqR(d).y
         P-control(s)
         == if P-control(y) then
              not Busy(y) and not Testing(y) and not Requesting(y)
            else false;
   PC3:  P-control(Complete.y)
         == if P-control(y) then Busy(y)
            else false;
   PC4:  P-control(TJ.y)
         == if P-control(y) then
              Busy(y) or Testing(y) or Requesting(y)
            else false;

   R1:   Requesting(null) == false;
   R2:   FOR s = ReqW(d,r).y AND ReqR(d).y
         Requesting(s) == if M-control(s) then true;
   R3:   Requesting(Complete.y)
         == if M-control(Complete.y) then Requesting(y);
   R4:   Requesting(TJ.y) == if M-control(TJ.y) then false;

   IPT1: Interpret(x,y)
         == if P-control(y) then
              x = Extract(y) and
              not Busy(y) and not Testing(y) and not Requesting(y)
            else false;

   EX1:  Extract(null) == null;
   EX2:  Extract(ReqW(d,r).y) == PUT(d,r).Extract(y);
   EX3:  Extract(ReqR(d).y) == GET(d).Extract(y);
   EX4:  FOR s = Complete.y AND TJ.y
         Extract(s) == Extract(y);

END;
```

図6. Spec Definition By Disksystem